



NASPA *technical*

Supporting Enterprise Networks and Operating Environments

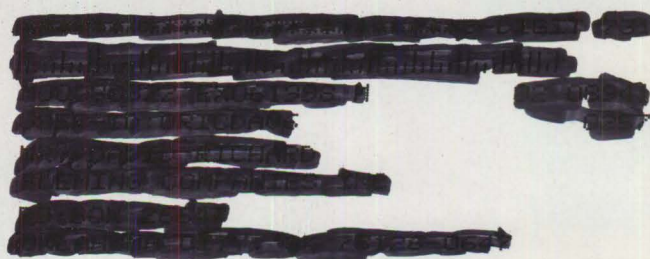
SUPPORT®

MARCH 1998 VOLUME 6, NUMBER 3

**Automated
File Transfer**

**Integrating
Windows NT
Into a NetWare
Environment**

www.naspa.net



FEATURES

10 Automated File Transfer Takes on DB2, Packed Fields, and the Year 2000

*By Charles A. Mills
and Mathew Bakulich*

File transfer can solve other problems besides simple *ad hoc* flat file downloads. If you need to transfer DB2 data to a server RDBMS, reformat fields for workstation processing, or fix the Year 2000

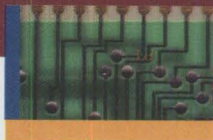


problem in downloaded data, you can find a file transfer solution that incorporates those features.

42 Integrating Windows NT Into an Existing NetWare Network: Part I

By John E. Johnston

It is very easy to set up a Windows NT network. However, with this ease of implementation comes a price. In fact, it is so simple to get a Windows NT network up and running that without realizing it, you could easily set up an extremely unmanageable, poorly performing network.



SYSTEMS

20 Vendor Performance Ratings: Why Your Results May Differ — Part IV

By Cheryl Watson

Your work may vary considerably from the workloads that were used by the vendor to determine the relative capacity and speed of a new model. It's up to you to determine how each of these factors will affect the real performance and capacity you receive.

24 Year 2000: Sorting Special Indicators With DFSORT's E61

By Frank L. Yaeger

Special indicators such as 000000 and 999999 mixed in with real dates can pose Year 2000 sorting problems. An E61 routine can be an effective way to handle these problems. By using the information DFSORT passes as well as information you read from a dataset, you can create flexible E61 routines to handle different century windows and all the types of dates you use.



NETWORKING

32 "Rewiring" Frame Relay: The Promise of ATM

By Leo A. Wrobel

The revolutionary changes occurring in the telecommunications environment represent not only cost-savings but also the perfect opportunity to expand the capabilities of your network.

48 Helping the Help Desk

By Steve Tielens

For many corporate help desks support is provided by one of the many new software tracking and monitoring solutions available. These powerful new tools quickly pinpoint software malfunctions, detect human interaction errors, and alert network managers and help desk personnel to individuals who might require further software training.

51 Protecting Your Networks With and Without Firewalls: Part VIII — NT Security

By Mark Bell

NT is becoming popular as both a file server and an application server. It installs TCP/IP as the default network protocol stack and contains many features that will encourage companies to use it in both intranet and Internet environments. While there has been a lot of marketing hype aimed toward reassuring administrators that NT is completely safe, as with any operating system you need to be aware of the risks associated with using it.



© 1997 Dynamic Graphics, Inc.

NaSPA Mission Statement:

The mission of NaSPA, Inc., a not-for-profit organization, shall be to serve as the means to enhance the status and promote the advancement of all network and systems professionals; nurture member's technical and managerial knowledge and skills; improve member's professional careers through the sharing and dispersing of technical information; promote the profession as a whole; further the understanding of the profession and foster understanding and respect for individuals within it; develop and improve educational standards; and assist in the continuing development of ethical standards for practitioners in the industry.

The information and articles in this magazine have not been subjected to any formal testing by NaSPA, Inc. or Technical Enterprises, Inc. The implementation, use and/or selection of software, hardware, or procedures presented within this publication and the results obtained from such selection or implementation, is the responsibility of the reader.

Articles and information will be presented as technically correct as possible, to the best knowledge of the author and editors. If the reader intends to make use of any of the information presented in this publication, please verify and test any and all procedures selected. Technical inaccuracies may arise from printing errors, new developments in the industry and/or changes or enhancements to components, either hardware or software.

The opinions expressed by the authors who contribute to *NaSPA Technical Support* are their own and do not necessarily reflect the official policy of NaSPA, Inc. Articles may be submitted by members of NaSPA, Inc. The articles should be within the scope of host-based, distributed platforms, network communications and data base, and should be a subject of interest to the members and based on the author's experience. Please call or write for more information. Upon publication, all letters, stories and articles become the property of NaSPA, Inc. and may be distributed to, and used by, all of its members.

NaSPA, Inc. is a not-for-profit, independent corporation and is not owned in whole or in part by any manufacturer of software or hardware. All corporate computing professionals are welcome to join NaSPA, Inc. For information on joining NaSPA and for membership rates, see page 73.

NaSPA Technical Support (ISSN 1079-3135) (IPM Agreement Number 0806773) is published monthly by Technical Enterprises Inc., 7044 S. 13th Street, Oak Creek, WI 53154-1429. Periodicals postage paid at Oak Creek, WI and additional mailing office. **POSTMASTER:** Send address changes to *NaSPA Technical Support*, 7044 S. 13th Street, Oak Creek, WI 53154-1429.

All product names mentioned in this publication are the trademarks/registered trademarks of their respective manufacturers.

C O L U M N S

- | | | | |
|-----------|---|-----------|--|
| 55 | MVS Tools & Tricks
Assorted Utilities: Part II
<i>By Sam Golob</i> | 66 | OS/2 Insights
Object-Oriented Programming
With Object REXX
<i>By Rick Byrley</i> |
| 58 | Storage Strategies
How Do Things Stack Up?
DFSMSHsm Tape Stacking
<i>By Steve Pryor</i> | 68 | NT Insights
NT Group and User
Management Strategies
<i>By Guy C. Yost</i> |
| 60 | Working Smarter
The DSLIST Utility of ISPF V4:
Part III
<i>By Jim Moore</i> | 72 | Evolutions
The Rich Diversity of Life
<i>By Michael Norton</i> |
| 62 | VSE Tools & Techniques
VSE Data Spaces: Part I
<i>By Leo J. Langevin</i> | 74 | Enterprise Networking
The Elusive Corporate Dial-up
Networking Solution
<i>By John E. Johnston</i> |
| 64 | Consultant's Corner
Jungle to Jungle
<i>By Jason Hannas</i> | 76 | On a Personal Note
The Folklore Factor
<i>By Mike Sutton</i> |

D E P A R T M E N T S

- | | | | |
|-----------|---------------------------------|-----------|-------------------------------------|
| 8 | Publisher's Letter | 50 | NaSPA Services Directory |
| 31 | HOTLINKS | 61 | NaSCOM Web Site Storage Info |
| 36 | NaSPA Direct | 70 | Education Vendors |
| 38 | NaSPA News | 73 | Reader Services |
| 39 | Letters to the Editor | 77 | Recent Releases |
| 45 | NaSPA Insurance Programs | | |

Object-Oriented Programming With Object REXX

BY RICK BYRLEY

In last month's column I demonstrated how to create an Object REXX "Hello World" program as an introduction to the Object REXX environment that comes with OS/2. As I mentioned, Object REXX is not only a useful programming language in its own right, but it is also an excellent introduction to object-oriented programming techniques for those used to the traditional, structured programming approach. Since "Classic REXX" programs will run under the Object REXX interpreter, REXX is perfect for experimenting with the two very different approaches and learning the principles of object-oriented programming. This month I will examine some of the fundamental concepts behind an object-oriented language and how these concepts apply to Object REXX in particular.

SOME PRELIMINARIES

Some very simple example code is provided to illustrate these principles, but first, a short course in Object REXX program structure is in order. There are two parts to any true object-oriented Object REXX program: the executable code and the class definitions. The executable code is placed at the beginning of the program and ends when the first directive is encountered. Typically the executable code will create instances of objects and then call the objects' various methods to perform the work of the program.

Directives (keywords preceded by two semicolons, e.g., ::CLASS) define classes and their methods. There are four directives and only two of them, ::CLASS and ::METHOD, will be of interest here. Note that the ::CLASS directive continues to define a class until the next ::CLASS directive is reached, and a ::METHOD directive continues to define a method until the next directive (either ::CLASS or ::METHOD) is encountered.

Now that these ground rules are out of the way, let's look at the three major principles of object-oriented programming: data encapsulation, polymorphism, and inheritance.

DATA ENCAPSULATION

There are many ways to define objects, but the simplest seems to be that an object is data and the code necessary to manipulate the data. Those of you familiar with BASIC programming might remember how data was stored within a program using the DATA statement, which made the task completely self-contained — it did not need to access data sources external to itself. I like to think of objects like those old BASIC programs: data and the code to manipulate that data. This illustrates one of the principles of object-oriented programming: data encapsulation. Each object has its own data that is protected from other objects.

An object is an abstraction defined by a class. Just like a program will do nothing until it is invoked, an object will do nothing unless an instance of the object is created. Once created — and this is important — according to the principle of data encapsulation each instance has its own copy of the object data.

For example, in the sample code "encapsulation.cmd" in Figure 1, two instances of MyClass are created, and the variable "MyVar" is set in each instance. The program then calls another method to display the value of the variable to illustrate that each instance contains its own version of the MyVar variable. This is data encapsulation.

POLYMORPHISM

As mentioned previously, an object is data and the code to manipulate that data. This code consists of object methods. A

Figure 1: ENCAPSULATION.CMD

```

/*****
*
* PROGRAM: ENCAPSULATION.CMD
* AUTHOR: Rick Byrley, SoftTouch Systems
* PURPOSE: Illustrates data encapsulation in Object REXX
*
*****/

/* Create two instances of the MyClass object */
instance1 = .MyClass-new
instance2 = .MyClass-new

/* Call each instance's method to demonstrate data encapsulation */
instance1~setMyVar("This is instance 1")
instance2~setMyVar("This is instance 2")

/* Display the value of the variable in each instance */
instance1~sayMyVar
instance2~sayMyVar

::class MyClass
::method 'setMyVar'
  expose myvar
  parse arg myvar
::method 'sayMyVar'
  expose myvar
  say myvar

```


Figure 2: POLYMORPHISM.CMD

```
/* *****
 *
 * PROGRAM: POLYMORPHISM.CMD
 * AUTHOR: Rick Byrley, SofTouch Systems
 * PURPOSE: Illustrates polymorphism in Object REXX
 *
 * ***** */

/* Create instances of the OneClass and AnotherClass objects */
instance1 = .OneClass~new
instance2 = .AnotherClass~new

/* Call each instance's method set the value of the variable */
instance1~setMyVar("This is instance 1")
instance2~setMyVar("This is instance 2")

/* Display the value of the variable in each instance */
instance1~sayMyVar
instance2~sayMyVar

::class OneClass
::method 'setMyVar'
  expose myvar
  parse arg myvar
::method 'sayMyVar'
  expose myvar
  say "This is OneClass MyVar:" myvar

::class AnotherClass
::method 'setMyVar'
  expose myvar
  parse arg myvar
::method 'sayMyVar'
  expose myvar
  say "This is AnotherClass MyVar:" myvar
```

Figure 3: INHERITANCE.CMD

```
/* *****
 *
 * PROGRAM: INHERITANCE.CMD
 * AUTHOR: Rick Byrley, SofTouch Systems
 * PURPOSE: Illustrates inheritance in Object REXX
 *
 * ***** */

/* Create instances of the OneClass and AnotherClass objects */
instance1 = .OneClass~new
instance2 = .AnotherClass~new

/* Call each instance's methods to say Hello */
instance1~sayHello
instance2~sayHello

::class AClass
::method sayHello
  say "Hello!"

::class OneClass subclass AClass

::class AnotherClass subclass AClass

::method sayHello
  say "Greetings!"
```

method is similar to a function in that it generally performs a discrete action. For example, a file object might have methods to read the file, write the file, copy the file, etc.

In contrast to traditional programming, methods do not carry the load of having to adapt within a single program to various kinds of data, since in an object the data is structurally identical each time.

"Reading" a text file is different than

"reading" the next item in a list, although both a file and list object may use a "read" method. This use of the same label to perform different actions on different objects is called polymorphism, and is another fundamental principle of object-oriented programming.

The sample code in Figure 2 illustrates polymorphism. The program creates two objects, one from the OneClass class and the other from the AnotherClass class.


Both classes have the identical methods setMyVar and sayMyVar. The first method is identical in both classes; sayMyVar, however, differs between the two classes.

INHERITANCE

Thus, each object has its own unique version of a method, more or less independent of all other objects and methods. I say more or less because a third principle of object-oriented programming, inheritance, determines which methods an object actually possesses. All objects (except for the object "object") inherit properties from another class. This feature is why object-oriented code is highly reusable: simply inherit the parts you want, override the parts you don't, code any unique parts, and you have a new object. Depending on which book you read, the class from which another class is derived is called a "parent" or "super." The super class is specified as an option to the ::CLASS directive when the class is defined by declaring the class to be a subclass of the named superclass.

The inheritance.cmd sample in Figure 3 is a simple case of inheritance. Both the OneClass and AnotherClass classes are subclasses of the AClass class. Therefore, they both inherit the "sayHello" method from the AClass class, which is why the method can be invoked on the OneClass instance (instance 1) although no such "sayHello" method is defined in the code for the class. The AnotherClass class overrides the default "sayHello" method defined by the parent class AClass, choosing to say "Greetings" rather than "Hello."

SUMMARY

Data encapsulation, polymorphism, and inheritance are the foundation of any object-oriented language, from Small Talk to C++ to Java to Object REXX. Fortunately, REXX is such an easy language syntactically that these principles aren't lost in the complexity of the code, which makes it a perfect introduction to object-oriented programming. And it's free with OS/2. 

Was this column of value to you? If so, please circle Reader Response Card No. 42.

Rick Byrley is senior workstation division technician for SofTouch Systems, Oklahoma City, Okla., which provides both mainframe and PC software solutions. His primary focus is object-oriented programming. He can be reached at rbyrley@softouch.com.